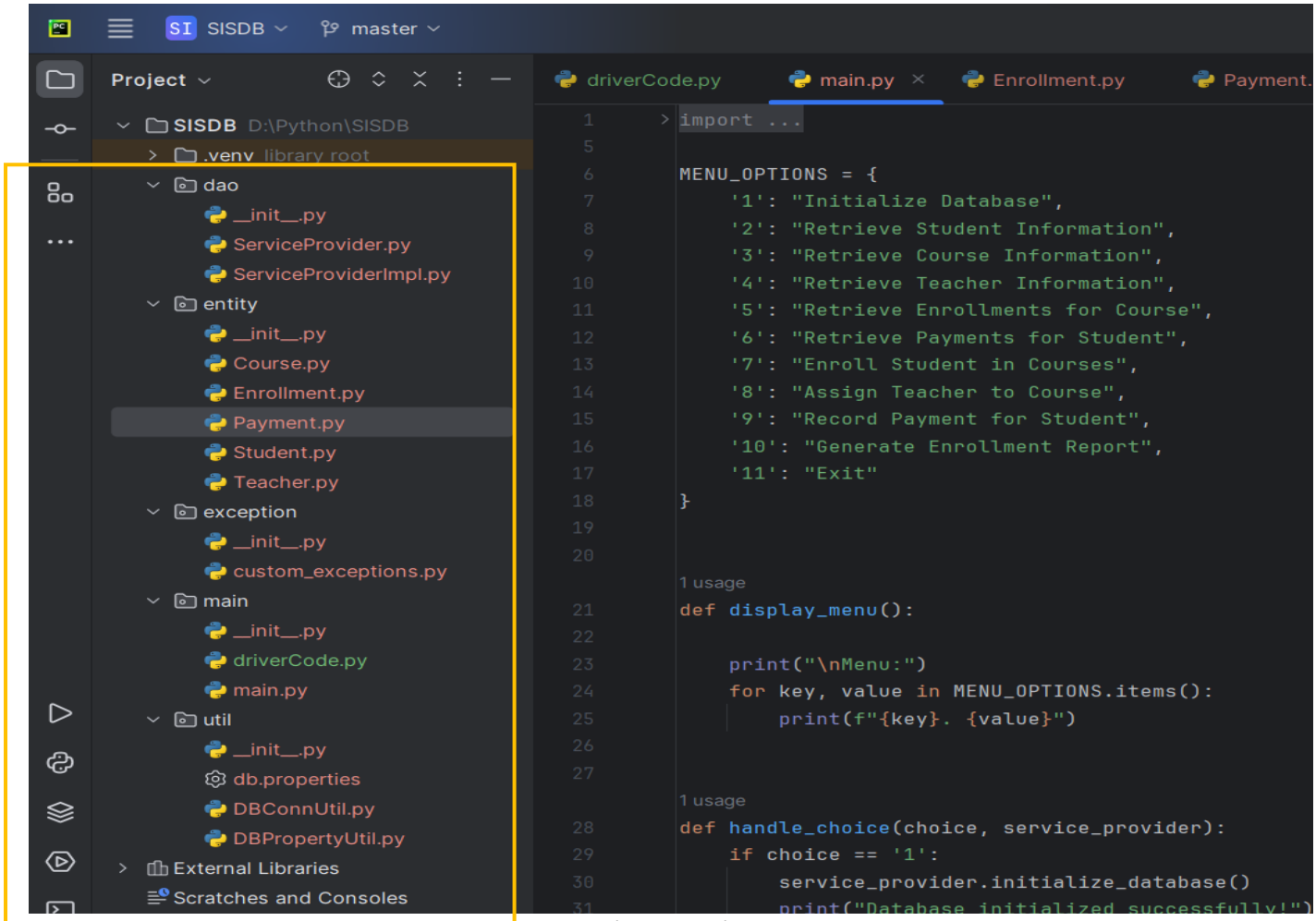
- The following Directory structure is to be followed in the application.
  - **entity/model**
    - Create entity classes in this package. All entity class should not have any business logic.
  - **dao**
    - Create Service Provider interface/abstract class to showcase functionalities.
    - Create the implementation class for the above interface/abstract class with db interaction.
  - **exception**
    - Create user defined exceptions in this package and handle exceptions whenever needed.
  - **util**
    - Create a DBPropertyUtil class with a static function which takes property file name as parameter and returns connection string.
    - Create a DBConnUtil class which holds static method which takes connection string as parameter file and returns connection object.
  - **main**
    - Create a class MainModule and demonstrate the functionalities in a menu driven application.



(Structure)

Task 1: Define Classes

Define the following classes based on the domain description:

Student class with the following attributes:

- Student ID
- First Name
- Last Name
- Date of Birth
- Email
- Phone Number

Course class with the following attributes:

- Course ID
- Course Name
- Course Code
- Instructor Name

Enrollment class to represent the relationship between students and courses.

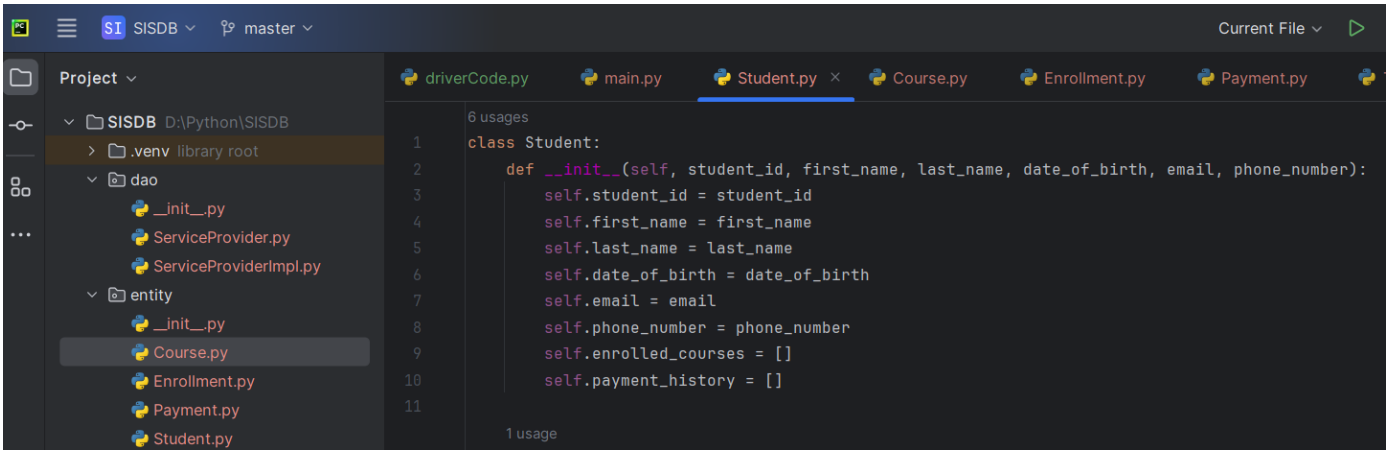
- Enrollment ID
- Student ID (reference to a Student)
- Course ID (reference to a Course)
- Enrollment Date

Teacher class with the following attributes:

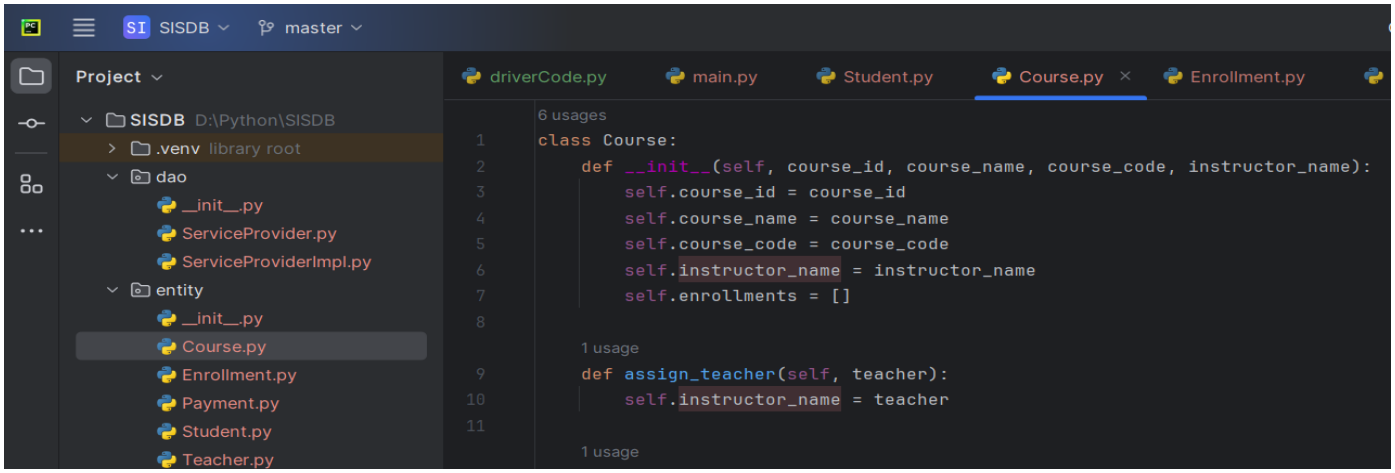
- Teacher ID
- First Name
- Last Name
- Email

Payment class with the following attributes:

- Payment ID
- Student ID (reference to a Student)
- Amount
- Payment Date



Student Class



Course Class

```
class Enrollment:
    def __init__(self, enrollment_id, student_id, course_id, enrollment_date):
        self.enrollment_id = enrollment_id
        self.student_id = student_id
        self.course_id = course_id
        self.enrollment_date = enrollment_date
```

Enrollment Class

```python
4 usages
class Teacher:
    def __init__(self, teacher_id, first_name, last_name, email):
        self.teacher_id = teacher_id
        self.first_name = first_name
        self.last_name = last_name
        self.email = email
        self.assigned_courses = []
```

Teacher Class

```python
4 usages
class Payment:
    def __init__(self, payment_id, student_id, amount, payment_date):
        self.payment_id = payment_id
        self.student_id = student_id
        self.amount = amount
        self.payment_date = payment_date
```

Payment Class

Task 3: Implement Methods

Implement methods in your classes to perform various operations related to the Student Information System (SIS). These methods will allow you to interact with and manipulate data within your system. Below are detailed instructions on how to implement methods in each class:

Implement the following methods in the appropriate classes:

Student Class:

- EnrollInCourse(course: Course): Enrolls the student in a course.
- UpdateStudentInfo(firstName: string, lastName: string, dateOfBirth: DateTime, email: string, phoneNumber: string): Updates the student's information.
- MakePayment(amount: decimal, paymentDate: DateTime): Records a payment made by the student.
- DisplayStudentInfo(): Displays detailed information about the student.
- GetEnrolledCourses(): Retrieves a list of courses in which the student is enrolled.
- GetPaymentHistory(): Retrieves a list of payment records for the student.

```python
1 usage
def enroll_in_course(self, course):
    self.enrolled_courses.append(course)

1 usage
def update_student_info(self, first_name, last_name, date_of_birth, email, phone_number):
    self.first_name = first_name
    self.last_name = last_name
    self.date_of_birth = date_of_birth
    self.email = email
    self.phone_number = phone_number

def make_payment(self, amount, payment_date):
    self.payment_history.append((amount, payment_date))
```

```
1 usage
def display_student_info(self):
    print(f"Student ID: {self.student_id}")
    print(f"Name: {self.first_name} {self.last_name}")
    print(f"Date of Birth: {self.date_of_birth}")
    print(f"Email: {self.email}")
    print(f"Phone Number: {self.phone_number}")

1 usage
def get_enrolled_courses(self):
    return self.enrolled_courses

3 usages
def get_payment_history(self):
    return self.payment_history

1 usage
def record_payment(self, amount):
    self.payment_history.append(amount)
```

Course Class:

- AssignTeacher(teacher: Teacher): Assigns a teacher to the course.
- UpdateCourseInfo(courseCode: string, courseName: string, instructor: string): Updates course information.
- DisplayCourseInfo(): Displays detailed information about the course.
- GetEnrollments(): Retrieves a list of student enrollments for the course.
- GetTeacher(): Retrieves the assigned teacher for the course.

```
1 usage
def assign_teacher(self, teacher):
    self.instructor_name = teacher

1 usage
def update_course_info(self, course_code, course_name, instructor):
    self.course_code = course_code
    self.course_name = course_name
    self.instructor_name = instructor

1 usage
def display_course_info(self):
    print(f"Course ID: {self.course_id}")
    print(f"Name: {self.course_name}")
    print(f"Code: {self.course_code}")
    print(f"Instructor: {self.instructor_name}")

3 usages
def get_enrollments(self):
    return self.enrollments

def get_teacher(self):
    return self.instructor_name
```

Enrollment Class:

- GetStudent(): Retrieves the student associated with the enrollment.
- GetCourse(): Retrieves the course associated with the enrollment.

```
3 usages (3 dynamic)
def get_student(self):
    return self.student_id

def get_course(self):
    return self.course_id
```

Teacher Class:

- UpdateTeacherInfo(name: string, email: string, expertise: string): Updates teacher information.
- DisplayTeacherInfo(): Displays detailed information about the teacher.
- GetAssignedCourses(): Retrieves a list of courses assigned to the teacher.

```
def update_teacher_info(self, name, email, expertise):
    self.first_name, self.last_name = name.split(' ')
    self.email = email

def display_teacher_info(self):
    print(f"Teacher ID: {self.teacher_id}")
    print(f"Name: {self.first_name} {self.last_name}")
    print(f"Email: {self.email}")

def get_assigned_courses(self):
    return self.assigned_courses
```

Payment Class:

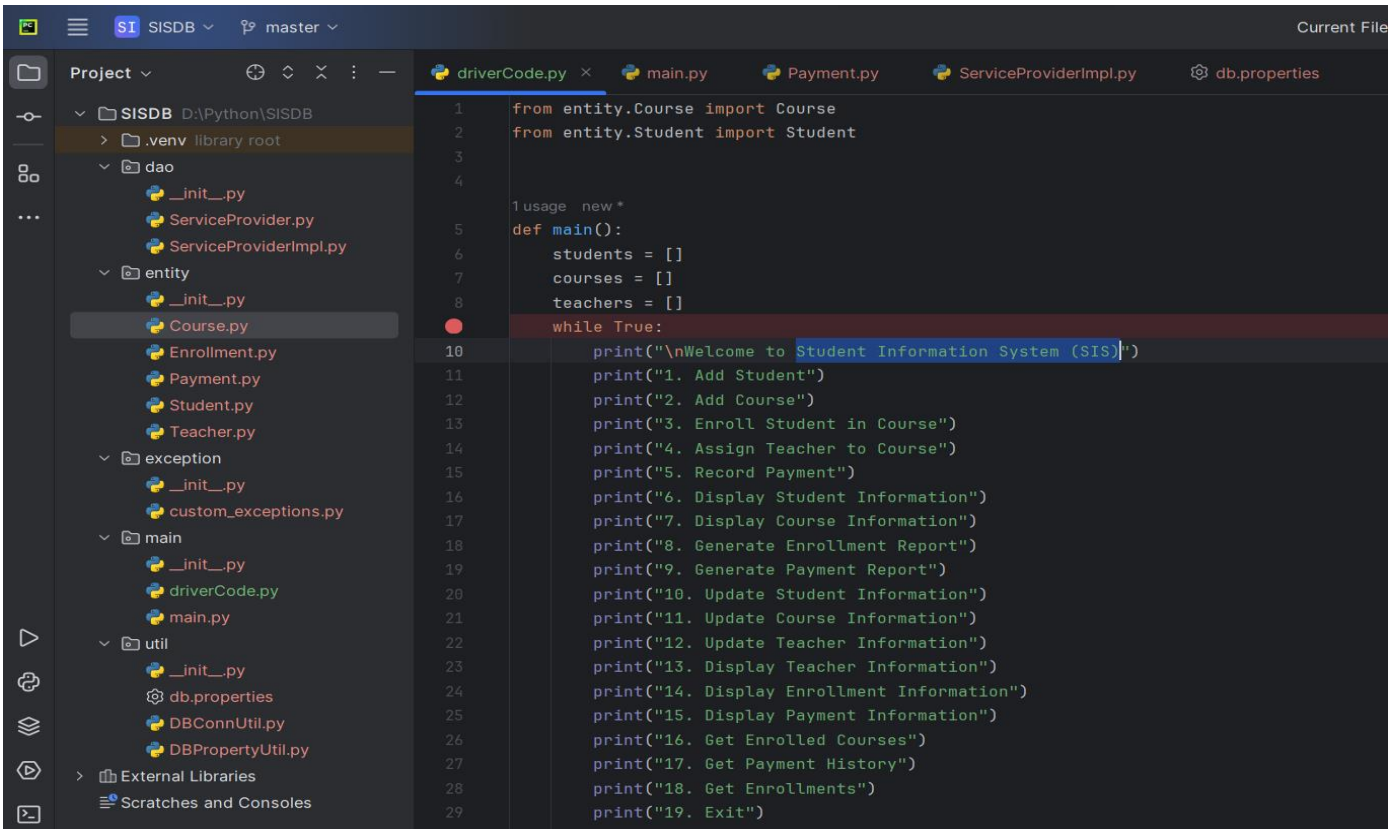
- GetStudent(): Retrieves the student associated with the payment.
- GetPaymentAmount(): Retrieves the payment amount.
- GetPaymentDate(): Retrieves the payment date.

```python
    3 usages (3 dynamic)
    def get_student(self):
        return self.student_id

    def get_payment_amount(self):
        return self.amount

    def get_payment_date(self):
        return self.payment_date

```

Implement the Main Method

In the console application, the Main method serves as the entry point for your program. This is where you will create instances of your classes, call methods, and interact with your Student Information System.

In the Main method, you create instances of your classes (e.g., Student, Course, and SIS) and then interact with your Student Information System by calling methods and handling exceptions.



Use the Methods

In your driver program or any part of your code where you want to perform actions related to the Student Information System, create instances of your classes, and use the methods you've implemented.

Repeat this process for using other methods you've implemented in your classes and the SIS class.

Few examples to show the Usage Of Implemented Methods:

Add Student:

```
Welcome to Student Information System (SIS)
1. Add Student
2. Add Course
3. Enroll Student in Course
4. Assign Teacher to Course
5. Record Payment
6. Display Student Information
7. Display Course Information
8. Generate Enrollment Report
9. Generate Payment Report
10. Update Student Information
11. Update Course Information
12. Update Teacher Information
13. Display Teacher Information
14. Display Enrollment Information
15. Display Payment Information
16. Get Enrolled Courses
17. Get Payment History
18. Get Enrollments
19. Exit
Enter your choice: 1
Enter Student ID: 1
Enter First Name: Biswarup
Enter Last Name: Roy
Enter Date of Birth (YYYY-MM-DD): 2000-03-27
Enter Email: roy_biswarup@yahoo.com
Enter Phone Number: 8694789197
Student added successfully!
```

Display Student Information:

```
Enter your choice: 6
Enter Student ID: 1
Student ID: 1
Name: Biswarup Roy
Date of Birth: 2000-03-27
Email: roy_biswarup@yahoo.com
Phone Number: 8694789197
```

Add Course:

```
19. Exit
Enter your choice: 2
Enter Course ID: 101
Enter Course Name: Mathematics
Enter Course Code: CS 101
Enter Instructor Name: Tanmoy Chakbraborty
Course added successfully!
```

Enroll in a course:

```
17. Get Payment History
18. Get Enrollments
19. Exit
Enter your choice: 3
Enter Student ID: 1
Enter Course ID: 101
Student enrolled in course successfully!
```

Payment to enroll in a course:

```
16. Get Enrolled Courses
17. Get Payment History
18. Get Enrollments
19. Exit
Enter your choice: 5
Enter Student ID: 1
Enter Payment Amount: 500
Payment recorded successfully!
```

Assign Teacher to a Course:

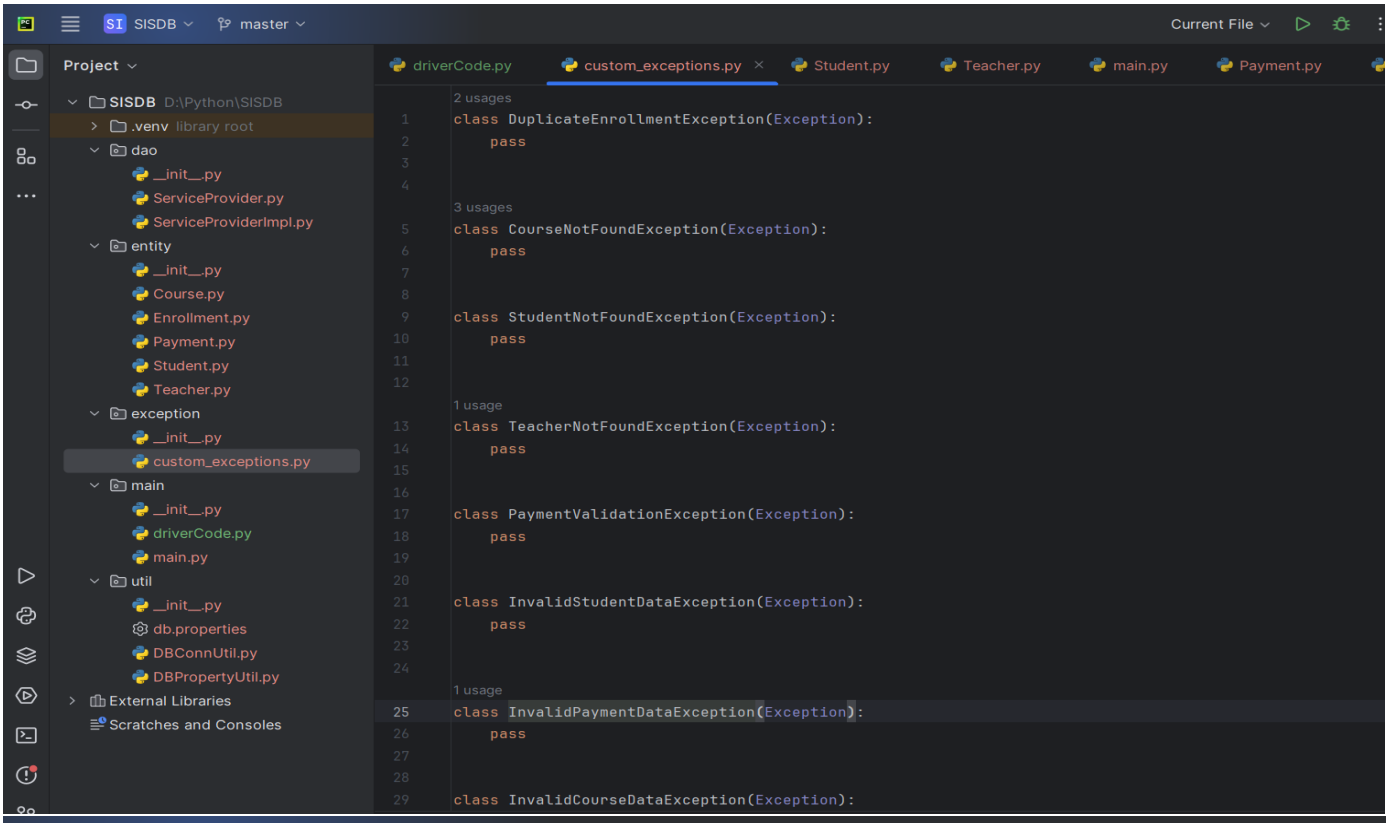
```
15. Display Payment Information
16. Get Enrolled Courses
17. Get Payment History
18. Get Enrollments
19. Exit
Enter your choice: Enter Course ID: 4
Enter Course Name: Mathematics
Enter Course Code: CS 101
Enter Instructor Name: Tanmoy Chakraborty
Course added successfully!
```

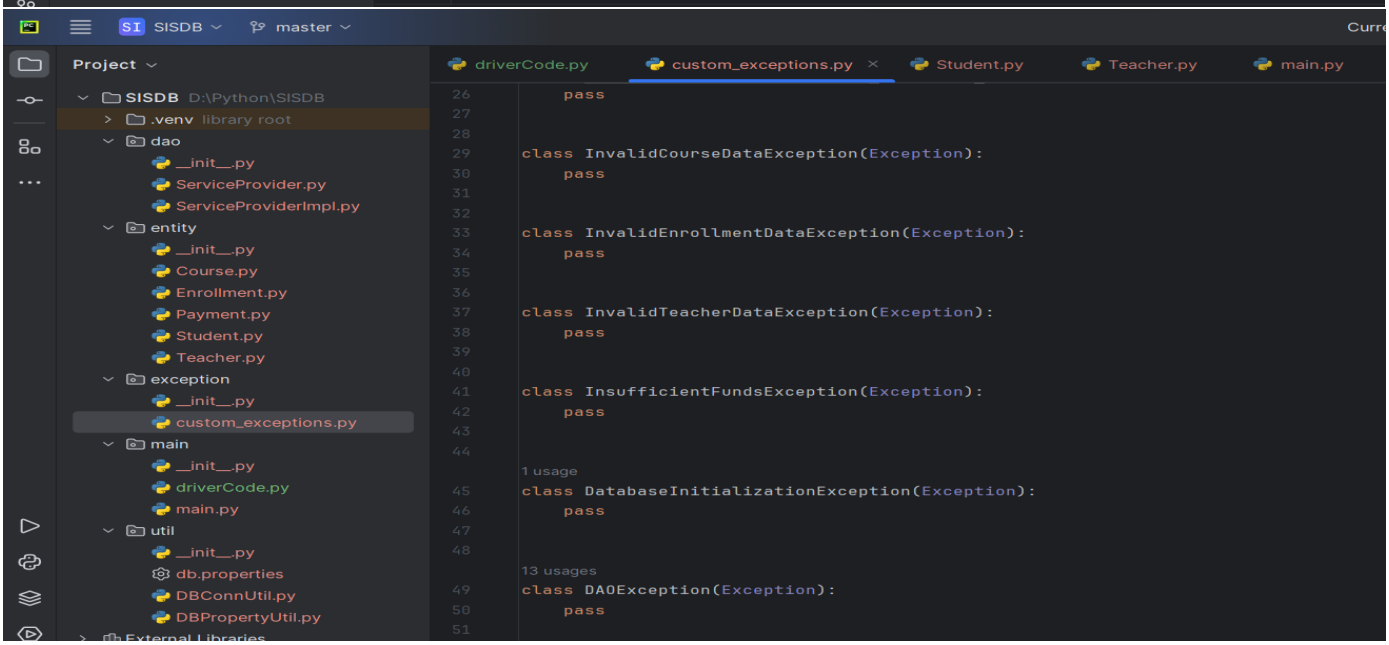
Task 4: Exceptions handling and Custom Exceptions

Implementing custom exceptions allows you to define and throw exceptions tailored to specific situations or business logic requirements.

Create Custom Exception Classes

You'll need to create custom exception classes that are inherited from the System.Exception class or one of its derived classes (e.g., System.ApplicationException). These custom exception classes will allow you to encapsulate specific error scenarios and provide meaningful error messages.





Task 7: Database Connectivity

Database Initialization:

Implement a method that initializes a database connection and creates tables for storing student, course, enrollment, teacher, and payment information. Create SQL scripts or use code-first migration to create tables with appropriate schemas for your SIS.

```python
    def initialize_database(self):
        try:
            cursor = self.connection.cursor()
            # Create tables if they don't exist
            cursor.execute("""
                CREATE TABLE IF NOT EXISTS Students (
                    student_id INT AUTO_INCREMENT PRIMARY KEY,
                    first_name VARCHAR(255),
                    last_name VARCHAR(255),
                    date_of_birth DATE,
                    email VARCHAR(255),
                    phone_number VARCHAR(20)
                )
            """)
            cursor.execute("""
                CREATE TABLE IF NOT EXISTS Courses (
                    course_id INT AUTO_INCREMENT PRIMARY KEY,
                    course_name VARCHAR(255),
                    credits INT,
                    teacher_id INT,
                    FOREIGN KEY (teacher_id) REFERENCES Teacher(teacher_id)
                )
            """)
            cursor.execute("""
                CREATE TABLE IF NOT EXISTS Enrollments (
                    enrollment_id INT AUTO_INCREMENT PRIMARY KEY,
                    student_id INT,
                    course_id INT,
                    enrollment_date DATE,
                    FOREIGN KEY (student_id) REFERENCES Students(student_id),
                    FOREIGN KEY (course_id) REFERENCES Courses(course_id)
                )
            """)
```

```python
            cursor.execute("""
                CREATE TABLE IF NOT EXISTS Teacher (
                    teacher_id INT AUTO_INCREMENT PRIMARY KEY,
                    first_name VARCHAR(255),
                    last_name VARCHAR(255),
                    email VARCHAR(255),
                )
            """)
            cursor.execute("""
                CREATE TABLE IF NOT EXISTS Payments (
                    payment_id INT AUTO_INCREMENT PRIMARY KEY,
                    student_id INT,
                    amount DECIMAL(10, 2),
                    payment_date DATE,
                    FOREIGN KEY (student_id) REFERENCES Students(student_id)
                )
            """)
            self.connection.commit()
            cursor.close()
        except mysql.connector.Error as e:
            raise DatabaseInitializationException(f"Error initializing database: {e}")
```

Data Retrieval:

Implement methods to retrieve data from the database. Users should be able to request information about students, courses, enrollments, teachers, or payments. Ensure that the data retrieval methods handle exceptions and edge cases gracefully.

```python
def retrieve_student_information(self, student_id: int):
    try:
        cursor = self.connection.cursor()
        cursor.execute("SELECT * FROM Students WHERE student_id = %s", (student_id,))
        student_data = cursor.fetchone()
        cursor.close()
        if student_data:
            return Student(*student_data)
        else:
            return None
    except mysql.connector.Error as e:
        raise DAOException(f"Error retrieving student information: {e}")
```

```python
def retrieve_course_information(self, course_id: int):
    try:
        cursor = self.connection.cursor()
        cursor.execute("SELECT * FROM Courses WHERE course_id = %s", (course_id,))
        course_data = cursor.fetchone()
        cursor.close()
        if course_data:
            return Course(*course_data)
        else:
            return None
    except mysql.connector.Error as e:
        raise DAOException(f"Error retrieving course information: {e}")
```

```python
1 usage (1 dynamic)
def retrieve_teacher_information(self, teacher_id: int):
    try:
        cursor = self.connection.cursor()
        cursor.execute("SELECT * FROM Teacher WHERE teacher_id = %s", (teacher_id,))
        teacher_data = cursor.fetchone()
        cursor.close()
        if teacher_data:
            return Teacher(*teacher_data)
        else:
            return None
    except mysql.connector.Error as e:
        raise DAOException(f"Error retrieving teacher information: {e}")
```

```
1 usage (1 dynamic)
def retrieve_enrollments_for_course(self, course_id: int):
    try:
        cursor = self.connection.cursor()
        cursor.execute("SELECT * FROM Enrollments WHERE course_id = %s", (course_id,))
        enrollment_data = cursor.fetchall()
        enrollments = [Enrollment(*enrollment) for enrollment in enrollment_data]
        cursor.close()
        return enrollments
    except mysql.connector.Error as e:
        raise DAOException(f"Error retrieving enrollments: {e}")

1 usage (1 dynamic)
def retrieve_payments_for_student(self, student_id: int):
    try:
        cursor = self.connection.cursor()
        cursor.execute("SELECT * FROM Payments WHERE student_id = %s", (student_id,))
        payment_data = cursor.fetchall()
        payments = [Payment(*payment) for payment in payment_data]
        cursor.close()
        return payments
    except mysql.connector.Error as e:
        raise DAOException(f"Error retrieving payments: {e}")
```

Data Insertion and Updating:

Implement methods to insert new data (e.g., enrollments, payments) into the database and update existing data (e.g., student information). Use methods to perform data insertion and updating. Implement validation checks to ensure data integrity and handle any errors during these operations.

```
1 usage (1 dynamic)
def enroll_student_in_courses(self, student_id: int, course_ids: List[int]):
    try:
        cursor = self.connection.cursor()
        for course_id in course_ids:
            cursor.execute("SELECT * FROM Enrollments WHERE student_id = %s AND course_id = %s",
                           (student_id, course_id))
            existing_enrollment = cursor.fetchone()
            if existing_enrollment:
                raise DuplicateEnrollmentException("Student is already enrolled in the course")

            # Check if the course exists
            cursor.execute("SELECT * FROM Courses WHERE course_id = %s", (course_id,))
            course_data = cursor.fetchone()
            if not course_data:
                raise CourseNotFoundException("Course not found")
            cursor.execute("""
                    INSERT INTO Enrollments (student_id, course_id, enrollment_date)
                    VALUES (%s, %s, %s)
                """, (student_id, course_id, datetime.now().date()))

        self.connection.commit()
        cursor.close()
    except mysql.connector.Error as e:
        raise DAOException(f"Error enrolling student in courses: {e}")
```

Transaction Management:

Implement methods for handling database transactions when enrolling students, assigning teachers, or recording payments. Transactions should be atomic and maintain data integrity. Use database transactions to ensure that multiple related operations either all succeed or all fail. Implement error handling and rollback mechanisms in case of transaction failures.

```
1 usage (1 dynamic)
def assign_teacher_to_course(self, course_id: int, teacher_id: int):
    try:
        cursor = self.connection.cursor()
        cursor.execute("UPDATE Courses SET teacher_id = %s WHERE course_id = %s", (teacher_id, course_id))
        self.connection.commit()
        cursor.close()
    except mysql.connector.Error as e:
        raise DAOException(f"Error assigning teacher to course: {e}")
```

Dynamic Query Builder:

Implement a dynamic query builder that allows users to construct and execute custom SQL queries to retrieve specific data from the database. Users should be able to specify columns, conditions, and sorting criteria. Create a query builder method that dynamically generates SQL queries based on user input. Implement parameterization and sanitation of user inputs to prevent SQL injection.

```
def assign_teacher_to_course(self, course_id: int, teacher_id: int):
    try:
        cursor = self.connection.cursor()
        cursor.execute("UPDATE Courses SET teacher_id = %s WHERE course_id = %s", (teacher_id, course_id))
        self.connection.commit()
        cursor.close()
    except mysql.connector.Error as e:
        raise DAOException(f"Error assigning teacher to course: {e}")

1 usage (1 dynamic)
def record_payment_for_student(self, student_id: int, amount: Decimal, payment_date: datetime):
    try:
        cursor = self.connection.cursor()
        cursor.execute("""
            INSERT INTO Payments (student_id, amount, payment_date)
            VALUES (%s, %s, %s)
        """, (student_id, amount, payment_date.date()))
        self.connection.commit()
        cursor.close()
    except mysql.connector.Error as e:
        raise DAOException(f"Error recording payment for student: {e}")
```

Task 8: Student Enrollment

In this task, a new student, John Doe, is enrolling in the SIS. The system needs to record John's information, including his personal details, and enroll him in a few courses. Database connectivity is required to store this information.

John Doe's details:

- First Name: John
- Last Name: Doe
- Date of Birth: 1995-08-15
- Email: john.doe@example.com
- Phone Number: 123-456-7890

John is enrolling in the following courses:

- Course 1: Introduction to Programming
- Course 2: Mathematics 101

The system should perform the following tasks:

- Create a new student record in the database.
- Enroll John in the specified courses by creating enrollment records in the database.

```
++++++++++++WELCOME TO Student Information System (SIS)++++++++++++

Menu:
1. Initialize Database
2. Retrieve Student Information
3. Retrieve Course Information
4. Retrieve Teacher Information
5. Retrieve Enrollments for Course
6. Retrieve Payments for Student
7. Enroll Student in Courses
8. Assign Teacher to Course
9. Record Payment for Student
10. Generate Enrollment Report
11. Add Student
12. Exit
Enter your choice: 11
Enter first name: John
Enter last name: Doe
Enter date of birth (YYYY-MM-DD): 1995-08-15
Enter email: john.doe@example.com
Enter phone number: 1234567890
Student added successfully!
```

```
select * from students;
select * from teacher;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Con

student_id	first_name	last_name	date_of_birth	email	phone_number
16	Zara	Ahmed	1998-12-30	zara.ahmed@example.com	4445556666
17	Vikram	Rahman	2000-07-08	vikram.rahman@example.com	9993337777
18	Ananya	Iqbal	1996-02-14	ananya.iqbal@example.com	8884441111
19	Rajesh	Islam	1999-11-05	rajesh.islam@example.com	6669992222
21	John	Doe	1995-08-15	john.doe@example.com	1234567890

```
Menu:
1. Initialize Database
2. Retrieve Student Information
3. Retrieve Course Information
4. Retrieve Teacher Information
5. Retrieve Enrollments for Course
6. Retrieve Payments for Student
7. Enroll Student in Courses
8. Assign Teacher to Course
9. Record Payment for Student
10. Generate Enrollment Report
11. Add Student
12. Exit
Enter your choice: 7
Enter Student ID: 22
Enter Course IDs (separated by comma): 1,7
Student enrolled in courses successfully!
```

```
Enter your choice: 10
Enter Course Name: Mathematics 101
Enrollment Report:
11, Biswarup, Roy, 2000-03-27, biswarup.roy@example.com, 8697841979, Mathematics 101
22, John, Doe, 1995-08-15, john.doe@example.com, 1234567890, Mathematics 101
```

Task 9: Teacher Assignment

In this task, a new teacher, Sarah Smith, is assigned to teach a course. The system needs to update the course record to reflect the teacher assignment.

Teacher's Details:

- Name: Sarah Smith
- Email: sarah.smith@example.com
- Expertise: Computer Science

Course to be assigned:

- Course Name: Advanced Database Management
- Course Code: CS302

The system should perform the following tasks:

- Retrieve the course record from the database based on the course code.
- Assign Sarah Smith as the instructor for the course.
- Update the course record in the database with the new instructor information.

```
Menu:
1. Initialize Database
2. Retrieve Student Information
3. Retrieve Course Information
4. Retrieve Teacher Information
5. Retrieve Enrollments for Course
6. Retrieve Payments for Student
7. Enroll Student in Courses
8. Assign Teacher to Course
9. Record Payment for Student
10. Generate Enrollment Report
11. Add Student
12. Exit
Enter your choice: 8
Enter Course ID: 11
Enter Teacher ID: 11
Teacher assigned to course successfully!
```

```
select * from courses;
select * from students;
select * from teacher;
```

Result Grid | Filter Rows: | Edit: | Export/Impor

course_id	course_name	credits	teacher_id
6	Chemistry	3	1
7	Introduction to Programming	4	2
8	Environmental Science	2	5
9	Economics Fundamentals	3	3
10	Hindi Literature	2	6
11	Advance Database Manage...	5	11
NULL	NULL	NULL	NULL

Task 10: Payment Record

In this task, a student, Jane Johnson, makes a payment for her enrolled courses. The system needs to record this payment in the database.

Jane Johnson's details:

- Student ID: 101
- Payment Amount: $500.00
- Payment Date: 2023-04-10

The system should perform the following tasks:

- Retrieve Jane Johnson's student record from the database based on her student ID.
- Record the payment information in the database, associating it with Jane's student record.
- Update Jane's outstanding balance in the database based on the payment amount.

```
1. Initialize Database
2. Retrieve Student Information
3. Retrieve Course Information
4. Retrieve Teacher Information
5. Retrieve Enrollments for Course
6. Retrieve Payments for Student
7. Enroll Student in Courses
8. Assign Teacher to Course
9. Record Payment for Student
10. Generate Enrollment Report
11. Add Student
12. Exit
Enter your choice: 9
Enter Student ID: 23
Enter Payment Amount: 500
Enter Payment Date (YYYY-MM-DD): 2023-04-10
Payment recorded successfully!
```

```
1. Initialize Database
2. Retrieve Student Information
3. Retrieve Course Information
4. Retrieve Teacher Information
5. Retrieve Enrollments for Course
6. Retrieve Payments for Student
7. Enroll Student in Courses
8. Assign Teacher to Course
9. Record Payment for Student
10. Generate Enrollment Report
11. Add Student
12. Exit
Enter your choice: 6
Enter Student ID: 23
Payments for Student:
Payment ID: 11 || Student ID: 23 || Amount: 500.0 || Payment Date: 2023-04-10
```

Task 11: Enrollment Report Generation

In this task, an administrator requests an enrollment report for a specific course, "Computer Science 101." The system needs to retrieve enrollment information from the database and generate a report.

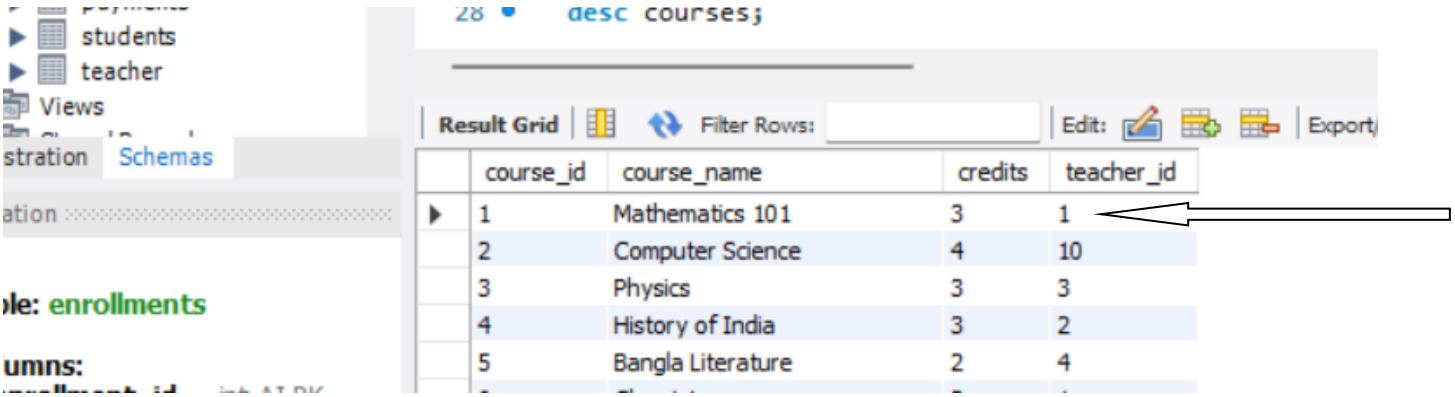
Course to generate the report for:

- Course Name: Computer Science 101

The system should perform the following tasks:

- Retrieve enrollment records from the database for the specified course.
- Generate an enrollment report listing all students enrolled in Computer Science 101.
- Display or save the report for the administrator.

```
Menu:
1. Initialize Database
2. Retrieve Student Information
3. Retrieve Course Information
4. Retrieve Teacher Information
5. Retrieve Enrollments for Course
6. Retrieve Payments for Student
7. Enroll Student in Courses
8. Assign Teacher to Course
9. Record Payment for Student
10. Generate Enrollment Report
11. Add Student
12. Exit
Enter your choice: 10
Enter Course Name: Mathematics 101
Enrollment Report:
11, Biswarup, Roy, 2000-03-27, biswarup.roy@example.com, 8697841979, Mathematics 101
22, John, Doe, 1995-08-15, john.doe@example.com, 1234567890, Mathematics 101
23, Jane, Johnson, 2000-03-27, johson_jane@example.com, 6547893214, Mathematics 101
```



course_id	course_name	credits	teacher_id
1	Mathematics 101	3	1
2	Computer Science	4	10
3	Physics	3	3
4	History of India	3	2
5	Bangla Literature	2	4

****END****